

Remarks

Status of application

Claims 1-61 were examined and stand finally rejected in view of prior art as well as for technical reasons. Applicant has amended claims 1 and 26 to address the Examiner's rejection of such claims under 35 U.S.C. Section 112 and requests entry of these claim amendments in order to reduce the issues for appeal. Additionally, Applicant has discussed the prior art rejections in detail and requests that the Examiner reconsider the prior art rejection of Applicant's claims for the reasons discussed below.

General

A. Objections to the Specification

The Examiner continues to object to Applicant's specification as containing hyperlinks stating for example that the reference in Applicant's specification to "eclipse.org" refers to a URL link on the Internet and therefore is improper. Applicant respectfully disagrees with the Examiner's position that paragraphs [0034], [0036] or other portions of Applicant's specification contain "hyperlinks" as defined in the MPEP as any URLs that are referenced in Applicant's specification are **not** preceded by "http://" (or "www") and are **not** placed between the symbols "< >". As MPEP § 608.01 provides that hyperlinks or a browser-executable code are a URL placed between the symbols "< >" or "http://" followed by a URL address, Applicant's specification does not include hyperlinks or browser-executable code. Although the MPEP prohibits use of hyperlinks or browser-executable code in the specification; it does not have any blanket prohibition against the mention of websites or URLs that are not hyperlinks such as, for example, "eclipse.org". As confirmation that this is the case, Applicant's representative performed a search for the term "Amazon.com" in published patents on the USPTO Patent Full Text and Image Database and found 429 hits (i.e., 429 patents including the term "Amazon.com" in the specification). Applicant therefore respectfully believes the Examiner's rejection is improper and should be withdrawn.

B. Section 112 Rejection

Claims 1-46 stand rejected under 35 U.S.C. Section 112, second paragraph as

being indefinite. Applicant has amended claims 1 and 26 to remove the reference to a "type Component", thereby overcoming the rejection.

Prior art rejections

A. First Section 103 rejection: DeGroot and McGurrin

Claims 1-5, 15-19, 21-26, 28-30, 33, 40-43, and 45-46 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,182,277 to DeGroot et al (hereinafter "DeGroot") in view of U.S. Patent 5,913,063 to McGurrin et al (hereinafter "McGurrin"). In the Final Rejection the Examiner continues to rely primarily on DeGroot's declarative programming techniques for object oriented environments as substantially equivalent to Applicant's attribute-based component programming system and methodology, but now acknowledges that DeGroot does not include teachings of generating a subclass based on a program class and at least one attribute, wherein the subclass includes dynamically generated program code based on such attribute(s) (Final Rejection, pages 4-5). Therefore, the Examiner adds McGurrin as providing these teachings.

However, turning to McGurrin one finds that its teachings are not at all analogous to Applicant's claimed invention. Although McGurrin does refer to an "attribute", the attribute described by McGurrin is something completely different from the attributes of Applicant's invention. McGurrin's attribute is also commonly known (e.g. in the Java programming language) as a "field" and is clearly described in this manner by McGurrin such as, for example, as follows:

In object oriented programming, the world is modeled in terms of objects. An object is a record combined with the procedures and functions that manipulate it. All objects in an object class have the same fields ("attributes"), and are manipulated by the same procedures and functions ("methods"). An object is said to be an "instance" of the object class to which it belongs. Sometimes an application requires the use of object classes that are similar, but not identical. For example, the object classes used to model both dolphins and dogs might include the attributes of nose, mouth, length and age. However, the dog object class may require a hair color attribute, while the dolphin object class requires a fin size attribute.

(McGurrin, col. 1, lines 10-25, emphasis added).

McGurrin goes on to describe that an object class may inherit attributes from another class. For example, both the "dog" and "dolphin" classes may inherit from an "animal" class so as to avoid having to duplicate code for the same attributes (e.g., nose, mouth, length, age) in multiple classes. McGurrin also describes the well-known technique of visually establishing relationships between attributes of objects in a visual programming environment. In other development environments, these attributes may be known as a "control" (if visual) or "instance variable" (if non-visual) rather than a field; however all of these are distinguishable from the attributes of Applicant's invention.

The attributes of Applicant's invention are defined as something completely different from those described by McGurrin. With Applicant's invention, the term "attributes" is defined as a language construct that programmers (developers) use to add additional information (i.e., metadata) to code elements (e.g., assemblies, modules, members, types, return values, and parameters) to extend their functionality (Applicant's specification, paragraph [0031]). These attributes are used by Applicant's invention to provide a flexible and extensible way of modifying the behavior of a program or a class or a method of a program (Applicant's specification, paragraph [0031]). A developer can utilize Applicant's attribute-based component programming system to define attributes that specify declaratively the addition of certain behavior to a program being developed. These attributes comprise "active" metadata used to generate code for adding behavior to a program as it executes at runtime. In response to the prior office action in this case, Applicant amended its claims to bring these teachings of attributes comprising "active" metadata for generating other code for inclusion in a program subclass to the forefront. For example, Applicant's claim 1 includes the following claim limitations:

A method for dynamically generating program code adding behavior to a program based on attributes, the method comprising:
adding a static field to a program class of the program to create a component;
defining at least one attribute specifying declaratively behavior to be added to the program, wherein said at least one attribute comprises active metadata used to generate program code for inclusion in the program;
associating said at least one attribute with the component; and
in response to instantiation of the component at runtime, generating a subclass based on the program class and said at least one attribute, the subclass including dynamically generated program code based on said at least one attribute.

(Applicant's amended claim 1, emphasis added)

With Applicant's claimed invention, attributes that are defined and added to a class or method result in additional code being automatically generated to implement the additional behavior (e.g., method tracing behavior) when the method is called at runtime (see e.g., Applicant's specification, paragraph [0084]). For instance, a developer may define attributes that specify the addition of method tracing behavior to a class being developed in order to assist in debugging the program that is being developed (see e.g., Applicant's specification, paragraph [0080], paragraph [0083]). For example, a developer may add the tracing attributes to an "add" method of a given class, so that when the "add" method is called, the defined attribute causes the tracing behavior to be added to the class. Thus, the "attributes" of Applicant's invention that comprise active metadata for generating code for inclusion in a program are fundamentally different both from the "fields" of McGurrian as well as from DeGroot's "rules" that define requirements or constraints.

Additionally, Applicant's claimed invention also provides for associating the defined attributes with a component so as to generate additional code at runtime for adding desired behavior to the program. This is described, for example, in Applicant's specification as follows:

The attributes added to the class typically result in the generation of extra code that is added into the dynamically generated subclasses. The Component class generates the subclass (i.e., creates the shell) for each of the components (i.e., component objects) that are instantiated. For each of the component objects in the class, the Component class essentially asks the attributes to specify the extra code that is to be inserted into the appropriate methods of the class.

(Applicant's specification, paragraph [0085], emphasis added).

Applicant's invention generates a subclass based on the original class and the attributes defined for the class. The attributes added to the class result in the generation of extra code that is added into the dynamically generated subclasses as described above. As discussed in detail in Applicant's previously filed Amendment, DeGroot does not include any comparable teachings generate additional code to extend program

functionality based on attributes. Although McGurrin describes subclassing in general terms, McGurrin does not teach dynamically generating program code based on defined attributes. Instead, McGurrin simply describes visual coding tools for generating code in response to user input (e.g., a user drawing a particular element on screen) as follows:

With a visual coding tool, some or all of the source code of a software application is generated automatically based on user manipulation of visual elements displayed on a computer screen. For example, many visual software development environments allow a programmer to "paint" a user interface by simply drawing user interface elements such as windows, buttons, and text fields on the screen. After drawing a user interface, the user presses a button to cause the visual coding tool to generate source code which, when compiled, produces executable code that generates the user interface. More sophisticated visual coding tools are capable of generating source code "on-the-fly" as the user draws the user interface.

Some visual coding tools generate source code for object oriented programming languages. With these visual coding tools, a generic object class is typically defined for every type of visual element. When a user draws a particular visual element, the visual coding tool generates source code for (1) a new object class that inherits from the corresponding generic object class, where the attributes of the subclass are initially set to those reflected in the element drawn by a user, and (2) an instance of the new object class.

For example, assume that a user draws a window that has a particular size, color and location within a visual coding environment. The visual coding tool will typically generate (1) source code that defines a subclass of a generic window class and sets the initial values of the size, color and location attributes of the subclass to values that correspond to the size color and location of window drawn by the user, and (2) source code that declares an instance of the new window subclass.

(McGurrin, col. 2, lines 14-45)

As shown above, with McGurrin's solution, code is generated in response to user interaction with a visual user interface of the program. For example, in response to a user drawing a particular element, code is generated in a subclass and attributes of the subclass and sets the initial size, color and location attributes of the subclass to the values corresponding to those drawn by the user. Thus, McGurrin is not adding code through the attributes but rather is providing data values for attributes that already exist as a part of a subclass. In contrast, Applicant's attributes specify declaratively behavior to be added to the program.

All told, the combined references provide no teaching or suggestion of defining attributes comprising active metadata and using these attributes for generating code for adding behavior to a program as it executes at runtime. Therefore, as DeGroot and McGurrin do not teach or suggest all of the claim limitations of Applicant's claims 1-5, 15-19, 21-26, 28-30, 33, 40-43, and 45-46 (and other claims) it is respectfully submitted that the claims distinguish over this reference and overcome any rejection under Section 103.

B. Second Section 103 rejection: DeGroot, McGurrin and Foster

Claims 6-14, 20, 31-39, and 44 stand rejected under 35 U.S.C. 103(a) as being unpatentable over DeGroot (above) in view of McGurrin (above) and further in view of U.S. Patent 7,103,885 of Foster (hereinafter "Foster").

Applicant's claims are believed to be allowable for at least the reasons cited above (as to the first Section 103 rejection) pertaining to the deficiencies of DeGroot and McGurrin as to Applicant's claimed invention. As these claims are dependent upon, and incorporate the limitations of Applicant's independent claims, they are distinguishable for the reasons previously described in detail. As Foster does not provide any teaching of defining attributes comprising active metadata used to generate code for adding behavior to a program as it executes at runtime, it does not cure the deficiencies of the DeGroot and McGurrin references as to Applicant's invention. In particular, although Foster does discuss "attributes" associated with source code files, the attributes described by Foster are represented by the presence (or not) of an attribute tag and an associated value in the comment field of a version management file (Foster col. 4, lines 10-13). As discussed in detail in Applicant's previously filed amendment, these attributes described by Foster are dramatically different than the attributes of Applicant's invention. Additionally, Foster includes no teaching of using the attributes as the basis for generating code for adding behavior to a software program as it executes at runtime.

All told, the combined references do not teach or suggest defining attributes which comprise active metadata and using such attributes for generating code for extending the functionality of a software program. Accordingly, as the combined references do not teach or suggest all of the limitation of Applicant's claims, it is

respectfully submitted that the claims distinguish over the combined references and overcome any rejection under Section 103.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

Applicant respectfully requests the entry of the amendments to the claims made in this Amendment After Final so as to narrow the grounds on Appeal. Applicant also requests the Examiner to reconsider the prior art rejections based on the remarks set forth herein.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 925 465-0361.

Respectfully submitted,

Date: December 9, 2008

/G. Mack Riddle/

G. Mack Riddle, Reg. No. 55,572
Attorney of Record

925 465-0361
925 465-8143 FAX